View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by ePublications at Regis University

Regis University
College for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

Use of the materials available in the Regis University Thesis Collection ("Collection") is limited and restricted to those users who agree to comply with the following terms of use. Regis University reserves the right to deny access to the Collection to any person who violates these terms of use or who seeks to or does alter, avoid or supersede the functional conditions, restrictions and limitations of the Collection.

The site may be used only for lawful purposes. The user is solely responsible for knowing and adhering to any and all applicable laws, rules, and regulations relating or pertaining to use of the Collection.

All content in this Collection is owned by and subject to the exclusive control of Regis University and the authors of the materials. It is available only for research purposes and may not be used in violation of copyright laws or for unlawful purposes. The materials may not be downloaded in whole or in part without permission of the copyright holder or as otherwise authorized in the "fair use" standards of the U.S. copyright laws and regulations.

PARTICIPATION IN THE ENVIRONMENTAL INFORMATION EXCHANGE NETWORK USING THE NATIONAL EMISSION INVENTORY DATAFLOW

Kent M. Thomas

Regis University

School For Professional Studies

Masters of Science in Computer Information Technology

Abstract

The US Environmental Protection Agency (EPA) and state environmental agencies have implemented a large nationwide system termed the Environmental Information Exchange Network, intended to consolidate and standardize the mechanism in which environmental data is exchanged between states, EPA, and other environmental organizations. The Exchange Network infrastructure is based on XML, Web services, and the Internet. The State of Alaska Department of Environmental Conservation (DEC) has an interest in participating in the Exchange Network. This project involves creation of software to extract, transform, validate, and submit DEC's air emissions data to EPA through the Exchange Network. Development of this software also represents a case study of the viability, benefits, and problems when transitioning an existing data exchange to a Service Oriented Architecture (SOA).

Acknowledgements

For Stephanie, Henry and Emma.

Many thanks to Dr. Doug Hart, Don Ina, and Joe Gerber for their valuable guidance and insights.

Contents

List of Tables

List of Figures

1. Chapter One: Introduction

1.1. Problem/opportunity to be addressed

Since humble beginnings in the mid 20th century, information technology (IT) has rapidly grown and spread throughout the world. With this rapid change and expansion, a complex and daunting array of IT technologies, platforms, tools, and techniques has emerged. As organizations increasingly rely on IT, the need to manage this complexity, gain efficiencies, and swiftly adapt to change is greater than ever. As the number of systems continues to grow, the need for standardized and efficient interactions between different systems has become increasingly prevalent, making application and data integration one of the foremost IT concerns today.

Vast quantities of environmental data are transferred between various entities in the US every day. Yet, prior to 2003, no national standard or infrastructure for exchanging environmental data existed, and many disparate mechanisms and formats emerged (Environmental Information Exchange Network, 2006). For example, water quality data has historically been submitted to the US Environmental Protection Agency (EPA) as an Oracle export file via file transfer, air emissions data as text files via a website upload, and facility enforcement data as text files via a mainframe upload. Similar varieties of exchange mechanisms have emerged for environmental data transfers not involving EPA. Custom data validation programs for each type of environmental data have also necessarily been built and maintained. The predictable result of this lack of standardization has been inefficient data exchange, poor data availability, poor data timeliness, fragile data integration, and high cost (Environmental Information Exchange Network, 2006).

As common focal points of environmental data transfer, EPA and state environmental agencies initiated a collaborative effort in 1998 to standardize environmental data flows. The Environmental Information Exchange Network (Exchange Network) is the culmination of this collaboration, promoting timely, cost-effective, and standardized environmental data exchange. Designed as a Service Oriented Architecture (SOA), and based on XML, Web services, and the Internet, the Exchange Network supports environmental data exchanges between diverse partners. With a formal launch in 2003, Exchange Network participation has gradually increased as support for specific environmental dataflows is added. To promote and facilitate a transition to the Exchange Network, EPA continues to provide funding to integrate partner systems. The State of Alaska Department of Environmental Conservation (DEC) is committed to participating in the Exchange Network, and has implemented a base Exchange Node, but no specific environmental dataflows have yet been developed.

All states must periodically submit National Emission Inventory (NEI) data to EPA, detailing emissions of air pollutants from various sources. This regulatory requirement has been in place for many years, preceding the existence of the Exchange Network. To date, DEC has fulfilled this requirement via a process that starts by exporting NEI data out of an Oracle database into a series of fixed length text files. These text files are validated using EPA's Basic Format and Content Checker utility, compressed, and manually uploaded using the EPA Central Data Exchange website (CDX Web). The current process to extract, transform, validate, and submit NEI data is slow, cumbersome, loosely integrated, and must be repeated for several NEI source data types. With the availability of the Exchange Network, an opportunity exists for

development of efficient and integrated software to process NEI data, as well as providing a basis for development of further Exchange Network dataflows.

1.2. Relevance, significance or need for the project

The Exchange Network offers a universal solution to the problem of disparate environmental data transfer mechanisms, supporting exchanges of all types of environmental data, via common data exchange definitions, XML, Web services, and the Internet. The potential benefits of using the Exchange Network are many, including reduced cost, improved data quality, flexibility to support new data exchanges, and access to real time data. These benefits are attractive to DEC, which is presently burdened with several disparate environmental data transfer mechanisms.

Although development of NEXT specifically applies to the Exchange Network, it also represents a case study in assessing the viability, benefits, and problems when transitioning an existing business process to an SOA implementation. Given that SOA has recently been the subject of considerable interest and debate in the general IT community, such a case study can provide useful insights for a broad spectrum of organizations considering use of an SOA.

1.3. Project goal

The Exchange Network provides a standardized and flexible framework for streamlining all environmental dataflows, one of which is the NEI. The goal of this project is to demonstrate the viability, benefits, and problems of transitioning an existing business process to an SOA

implementation, by developing software that implements an Exchange Network NEI dataflow from DEC to EPA. To this end, this project will involve development of a Microsoft C# program and set of supporting Oracle stored procedures, collectively named the NEI Exchange Toolkit (NEXT). NEXT will extract and transform NEI data into the NEI XML format. Resulting XML data will be validated using appropriate schema languages, and submitted to EPA's CDX Test Node (EPA's point of presence on the Exchange Network) through a series of published Web services. NEXT will perform the extract, transform, validate, and submit functions in an automated fashion, requiring minimal user interaction.

1.4. Barriers and/or issues (risks)

1.4.1. Forthcoming NEI regulation changes

The Consolidated Emission Reporting Rule (CERR) mandates periodic submission of air emission inventory data to EPA by states and a few other environmental entities. The most recent version of this rule was enacted in 2002. In 2006, EPA proposed an updated version of the CERR, where the primary change is shortened reporting timeframes. As of this writing, proposed changes do not appear to affect NEI data elements or the mechanics of the NEI Exchange Network dataflow. However, the proposed rule has not yet been finalized or formally enacted.

Global warming has become a front-line issue with many policy makers and the public, and assessing emissions of greenhouse gases is a corresponding concern. While greenhouse gases are not pollutants in the traditional sense, the NEI can be used to store and report greenhouse gas

emissions. Given this capable and existing infrastructure, NEI regulations may be updated in the near future to require the inclusion of greenhouse gas emissions.

1.4.2. Forthcoming NEI data specification changes

Prior to the implementation of the Exchange Network, state NEI data submissions to EPA were formatted according to the National Emission Inventory Input Format (NIF), which represents a proprietary text or Microsoft Access file format. The most recent version of the NIF is version 3.0, released in 2003. The NIF is still an acceptable format for states that have not yet implemented an NEI Exchange Network dataflow. In 2005, a NIF version 4.0 was in development, which included significant changes from prior versions. However, in 2006, development of version 4.0 was suspended to allow states to focus on implementing an NEI dataflow in the Exchange Network. While the Exchange Network uses XML and not the NIF, the NEI XML schemas are closely based on the NIF. As such, when NIF 4.0 is released, corresponding changes will also be implemented to the NEI XML schemas. While EPA has not announced a specific date in which NIF 4.0 will be introduced, this will likely occur in the near future.

1.4.3. Forthcoming Exchange Network infrastructure technology changes

Intended for implementation in 2008, the Node 2.0 specification updates the Exchange Network infrastructure to utilize current Web services standards. These changes are primarily intended to make development of Exchange Network software easier, and enable continued support from

software vendors. The specification is not finalized yet, but the latest specification draft includes changes to Web service call mechanisms, which will require NEXT coding alterations.

1.4.4. Large Alaska NEI dataset

The dataset that comprises Alaska's emission inventory for a given year is substantial, containing over 15,000 data records. It is not presently known to what extent DEC's existing NEI data meets the Exchange Network's XML validation criteria. If a sizeable number of validation errors are discovered during this project, it may be difficult to achieve a fully validated dataset, given an aggressive completion timeframe. Because an invalid dataset cannot be submitted to the Exchange Network, if an excessive number of validation errors occur, use of a small subset of test data will be necessary. However, using a small subset of test data would leave open the possibility of program anomalies in processing data outside this subset.

1.4.5. Complex project; uncharted territory for DEC

This project involves a complex set of data elements, transformations, and remote Web service references. Most other states have hired contractors to perform Exchange Network-related development, and most often using one of a handful of firms that specialize in this area. The NEI dataflow software will be developed by the author, and represents DEC's first Exchange Network dataflow. While the process of participating in the Exchange Network is well documented, this project represents new territory for DEC. Because of this uncertainty, unforeseen issues or project delays may occur.

1.5. Elements, hypotheses, theories, or questions to be discussed / answered

NEI data is sent through the Exchange Network via Web service calls. Since the amount of data to be sent can be substantial, the ability of Web services to be able to handle such calls is a concern (MacDonald, 2003). However, states with far larger datasets than Alaska are using the Exchange Network, and it is presumed that this consideration has been sufficiently addressed. The exact mechanism to handle large Web service calls in a robust manner is not known at the outset of this project, but mechanisms such as compression or asynchronous calls may be utilized.

A key aspect of this project is writing NEI data to an XML format. Most modern development tools and databases provide features for working with XML, including both reading and writing. At DEC, the software development environment is Microsoft Visual Studio 2005 and C#, and the database that stores the source NEI data is Oracle 10g. Either C# or Oracle PL/SQL can write to XML. Determining the optimal language in which to write XML is a fundamental design decision for this project. Determining how and where XML should be validated is an equally important design consideration.

Given the large amount of NEI data and anticipated processing involved, a set of Oracle PL/SQL stored procedures is likely necessary to ensure optimal performance in extracting appropriate NEI data (Feuerstein & Pribyl, 1997). Using C# and ADO.NET to perform many calls to the database would generally follow the same design pattern as DEC's legacy non-XML NEI extract program (see 2.1.7), and would thus likely result in poor performance.

1.6. Limitations/scope of the project

1.6.1. Test submittal only

Formal NEI submittals are due to the EPA CDX by June 1 of each year (either the CDX Web for non-Exchange Network submittals, or the CDX Production Node for Exchange Network submittals). The formal submittal is preceded by a substantial data entry effort at DEC to obtain air emissions source data from various sources. Given the significance of this effort, the formal NEI submittal has historically occurred just before June 1. Use of the Exchange Network only addresses submission of NEI data that is already obtained and stored in state databases— it does not address the time involved in obtaining emission data from industry to populate source databases. Because this project will start in early August, and conclude by October 21, a production NEI submittal cannot occur. Nonetheless, a successful submittal to EPA's CDX Test Node will occur as part of this project.

1.6.2. NEI dataflow only

The Exchange Network supports many types of environmental dataflows, and the number of supported dataflows is anticipated to increase in the future as EPA alters its many legacy systems to utilize the Exchange Network (Environmental Information Exchange Network, 2002). While DEC does not currently participate in any dataflow, DEC is considering implementing the dataflows noted in Table 1.

Table 1. Planned DEC dataflows

ID	Description
NEI	National emission inventory
FRS	Facility registry system
ICIS – NPDES	Water quality and discharge data
SDWIS	Safe drinking water info system

While the dataflows noted in Table 1 are currently supported by the Exchange Network, DEC must implement them in order to use them. Development of NEXT will represent DEC's first dataflow to the Exchange Network, and will implement the NEI dataflow only.

1.6.3. No modifications to source database

The DEC data center houses two Oracle 10g database servers, one for production, and another for development. For this project, the NEI data in the development database will be synchronized with the production database to ensure that a full and complete dataset is used. This project will exclusively work with the Oracle development database. The NEI data stored in DEC's Oracle databases is directly used by DEC's existing AirTools application, for management of air permits and other functions. Because the database is tightly coupled to AirTools, the existing database structure will not be altered in any way as part of this project. This project is limited to extracting data from this existing AirTools database, transforming to XML, validating, and submitting to EPA's CDX Test Node.

1.6.4. Node client only

This project will involve creation of a Node Client only, which represents an independent program that can connect to Exchange Network Nodes. While DEC has an installed Exchange Node where NEXT functionality could be integrated, such an approach is problematic due to the complexity involved and lack of available support by the contractor that built DEC's Node. Moreover, DEC has a regulatory requirement to submit NEI data to EPA annually, which can be wholly fulfilled with a Node Client. Use of the DEC Node would primarily serve to publish NEI data to other parties, or retrieve NEI data from other states. Yet DEC has no present knowledge of a need for such optional services. Integration of the NEI dataflow into the Exchange Node may occur in a future evolution of NEXT.

1.7. Definition of terms and acronyms

Term/acronym	Definition
ADO.NET	A set of data access components incorporated into the Microsoft .NET Framework, used to query and manipulate data in a variety of data sources (most often a relational database).
BCL	.NET Base Class Library. A library of .NET classes available to all .NET languages, performing common programming tasks such as file reading, file writing, rendering, database interaction, etc.
BFCC	Basic Format and Content Checker. A Visual Basic 6 utility

	program provided by EPA to validate NIF data files (either text or Microsoft Access).
C#	Microsoft's C# Programming Language. A general purpose, object-oriented programming language introduced in 2001, particularly suited to developing Windows and web applications based on the .NET Framework.
CDX web	EPA Central Data Exchange web portal. A website that acts as EPA's central "gateway" for environmental data submittals (non-Exchange Network based).
CDX node	EPA Exchange Network Node. EPA's central point of presence on the Exchange Network. Both a test and production node are available.
CERR	Consolidated Emission Reporting Rule. The federal law that mandates submission of emission inventory data to EPA, and specifies what data must be included and when it must be submitted (40 CFR 51, subpart A, and 40 CFR 51.122).
DEC	Alaska Department of Environmental Conservation. As the Alaska state environmental agency, DEC is tasked to "Conserve, improve, and protect Alaska's natural resources and the environment and control water, land, and air pollution, in order to enhance the health, safety, and welfare of the people of the state and their overall economic and social well being."
Dataflow	A type of environmental data that can exchanged on the

	Exchange Network between two or more partners (also termed a "Data Exchange.") The dataflow is defined using a Flow Configuration Document (FCD), XML Schema, and Data Exchange Template (DET).
DET	Data Exchange Template. A document that outlines the XML Schema for a particular dataflow, with validation rules and example content. The purpose of this template is to provide a more human readable version of an XML Schema.
DIME	Direct Internet Message Encapsulation. A mechanism for including binary attachments to Web service calls.
EPA	U.S. Environmental Protection Agency.
Environmental Information Exchange Network (aka Exchange Network)	An XML and Web services network, intended to streamline and standardize the mechanism in which environmental data is transferred between states, EPA, and other environmental organizations.
FCD	Flow Configuration Document. A document that details the rules governing a particular Exchange Network dataflow, using text, diagrams, and examples.
MTOM	Message Transmission Optimization Mechanism. A mechanism for including binary attachments to Web Service calls.
NAAS	Network Authentication and Authorization Service. A centralized service that maintains a list of valid Exchange

	Network users and their associated privileges.
Network node (aka Node)	A Web services-based server maintained by Exchange Network partners, responding to requests from other Nodes and submitting data to other Nodes.
Node client	A service or application that communicates directly with a Node.
NEI	National Emission Inventory. An EPA database that tracks emissions of various air pollutants around the nation. In a Exchange Network context, represents the dataflow that states use to populate this EPA database.
NEXT	NEI Exchange Toolkit. The name of the software created in this project, consisting of both C# and PL/SQL code. This software will extract, transform, validate, and submit DEC's air emissions data to EPA using the NEI dataflow.
Oracle RDBMS	Oracle relational database management system.
PL/SQL	Oracle Procedural Language/Structured Query Language. Oracle's extension to the SQL language, providing procedural programming constructs. PL/SQL is commonly used to write Oracle stored procedures and triggers.
Schematron	A type of XML schema language that extends the validation that can be performed by other schema languages.
SOA	Service Oriented Architecture. A flexible and adaptable software architecture based on loosely coupled services, descriptions, and messages.

SOAP	A standard protocol for transmitting XML messages when using Web services.
SQL	Structured Query Language. A language that consists of commands for querying and manipulating data and objects in a relational database.
Trading partner	An organization with an Exchange Network Node that is able to exchange data with another partner on the Exchange Network.
Web service	A web Application Programming Interface (API) that communicates using XML messages.
W3C	World Wide Web Consortium. The primary international standards body for the World Wide Web.
WSDL	Web services description language. Provides a model for describing Web services.
XML	Extensible Markup Language. An extensible, user-definable data format that is typically used to facilitate data exchange between heterogeneous systems.
XML Schema	W3C XML Schema. A formal definition of the required structure and format of a particular XML document.

1.8. Summary

IT is growing increasing complex, and one of the foremost issues facing IT today is application and data integration. The Exchange Network is an ambitious effort to standardize exchange of environmental data nationwide. While still evolving, the Exchange Network offers a host of

potential benefits today, including reduced cost, more timely data, higher quality data, and greater interoperability. This project will evaluate the viability, benefits, and problems of participating in an SOA implementation. As a case study, software named NEXT will be created to extract NEI data from an existing DEC database, transform this data as XML, validate the XML, and finally submit the XML to the CDX Test Node. Changes to NEI regulations, data formats, and the Exchange Network infrastructure are forthcoming, which introduces uncertainty in this project. This project will involve creation of a Node Client only, utilizing DEC's development Oracle database.

2. Chapter Two: Review of Literature / Research

2.1. Literature and research that is specific / relevant to the project

2.1.1. Service Oriented Architecture

In a Service Oriented Architecture (SOA), complex enterprise systems are decomposed into smaller, logical building blocks as a means to enhance flexibility, adaptability, and interoperability. Each individual building block provides a logical encapsulation of some unit of work, typically representing certain business logic. Building blocks are fully autonomous and self-contained, and may be invoked by other programs or other building blocks. Such building blocks are known as *services*. Services are formally described using standardized service *descriptions,* which minimally describe the service identity (name), data expected, and data returned. Services communicate using standardized *messages*. An SOA must adhere to several design principles when shaping services, descriptions, and messages (Erl, 2005), as noted in Table 2. An SOA paradigm can offer several benefits, as noted in Table 3. Most SOA benefits will not manifest themselves fully until SOA principles become established within the SOA implementation context, however (Erl, 2005).

Table 2. SOA design principles

Principle	**Description**
Loose coupling	Service-to-service and program-to-service dependencies are minimized.
Contract	Services adhere to a standardized and technology-independent communication agreement (i.e. interface), as specified in service descriptions.

Discoverability	Services can be found by potential requesters.
Reuse	Services are designed to promote reuse by requesters.
Abstraction	Only service interfaces are exposed to requesters, whereas the internal service logic (i.e. implementation) is hidden.
Autonomy	Services are independent, maintaining control over the logic they encapsulate.
Statelessness	Service communications minimize retention of information specific to an activity.
Aggregation	Collections of services can be assembled to form composite services.

Table 3. SOA benefits

Benefit	**Description**
Improved integration	Application integration is less costly and more efficient, due to intrinsic consistency and interoperability.
Enhanced solution architectures	Automation, consistency, and reduced processing requirements reduce cost and increase efficiency.
Leverage existing assets	Business logic in existing applications and systems can be exposed using services.
Inherent reuse	Services are designed for reuse, reducing the cost and effort of building solutions (although initial development effort is increased).
Standardized data representation	Standards-based data representation facilitates interoperability, reducing cost and increasing efficiency.

Improved organizational agility	The cost and effort to adapt and respond to business or technology changes is reduced.

2.1.2. XML

Markup languages are used to combine text and text descriptions, and include Standardized General Markup Language (SGML), Hypertext Markup Language (HTML), Extensible Markup Language (XML), Extensible Hypertext Markup Language (XHTML), and others. XML is a general-purpose markup language that is extensible because it supports creation of custom tags (Ray, 2003). XML carries data (or content) between custom-defined tags, specified within brackets. Every element with some data must contain a start and end tag. In the XML example noted in Figure 1, street is one *element,* where the street element tags (<street> and </street>) surround the content (123 Main St.).

Figure 1. XML address example

```
<?xml version="1.0"?>
<address>
   <street>123 Main St.</street>
   <city>Anchorage</city>
   <state>AK</city>
   <zip>99516</zip>
</address>
```

Developed by the World Wide Web Consortium (W3C) in the mid 1990s, XML has become the de-facto standard for data exchange over the Internet, primarily due to its wide acceptance and flexibility (Ray, 2003). Like HTML, XML emerged from SGML, as a smaller, leaner language well suited to the bandwidth sensitive Internet. Unlike HTML, XML predefines no tags, and has strict syntax requirements.

An XML *document* consists of a set of XML elements and other markup together in a package. An XML document exhibits two levels of integrity: *well formed* and *valid*. An XML document is well formed if all elements have proper starting and ending tags, and some other basic syntax rules are followed. A valid check is stricter than well-formed check, and provides further quality control assurance for the XML document. An XML document is considered valid if its structure and elements conform to a particular specification, which is defined using a particular *schema language*. Several different types of schema languages exist, with varying advantages and disadvantages. The Document Type Definition (DTD) contains a collection of rules that define elements and other markup objects. The XML Schema extends the DTD, by allowing specification of valid document content and data patterns. RELAX NG specifies patterns that define the structure of an XML document, using simple and elegant syntax. Schematron is a general and flexible schema language that uses XPath to reach portions of source XML documents. Schematron is of limited value by itself, but is powerful when used to augment another schema language.

XML can store documents (such as word processing documents), or data. XML is well suited to store diverse types of data, although it works best for small data sets and data that only needs to be sequentially searched. XML can additionally be transformed to present data (in a format such as HTML), when coupled with a stylesheet (CSS or XSL). XML is not a panacea, however. An oft-cited problem with XML is its generally large file size, when compared with binary formats that store the same information. For large XML files, traditional file compression tools such as ZIP or GZIP are sometimes used to reduce the file size.

2.1.3. Web services

Middleware is software that connects (or integrates) applications or components, most often between different machines distributed on a network (Britton & Bye, 2004). Many different types of middleware exist, including Remote Procedure Call (RPC), Distributed Component Object Model (DCOM), Common Object Request Broker Architecture (CORBA), Enterprise JavaBeans (EJB), messages queues, and various remote database access technologies. Web services are a newer form of middleware unlike other types in their use of open standards and decoupling from particular languages, platforms, and vendors. Several core specifications define Web services, including XML, SOAP, WSDL, and UDDI; these specifications are maintained by the W3C.

Web services communicate using the SOAP protocol. Typically working with either HTTP or HTTPS as the transport mechanism, SOAP defines an XML envelope that contains requests to and responses from a Web service. Use of SOAP and XML abstracts Web service

implementation and deployment technologies. For example, a Java program running on a Solaris UNIX box can call a Web service on a Windows box that was written in VB.NET.

To allow unambiguous use by a service requester, distributed computing services must provide a formal service description, generically created using an Interface Definition Language (IDL). The syntax associated with invocation and response of a Web service is described using Web Services Description Language (WSDL). In particular, WSDL defines what the service does, how the service is accessed, and where the service is located. WSDL is itself an XML document that conforms to the WSDL XML Schema.

The Universal Description, Discovery, and Integration (UDDI) protocol provides a standard, interoperable way for Web services to be advertised, discovered, and searched by potential requesters. UDDI allows the creation of different registries, which can serve different purposes in different contexts (for example, a UDDI registry might be created for Web services related to automotive repair). Using UDDI registries, service providers can advertise their services, and potential service requesters can search registries for areas of interest. While conceptually useful, UDDI registries have generally not yet achieved widespread usage.

Due to their design characteristics, Web services are well suited to implement an SOA (use of Web services does not necessarily result in a true SOA implementation, however). Using HTTP as a transport protocol, Web services are also well suited for use over the ubiquitous Internet.

Use of HTTP enables free flow of Web services traffic through most firewalls, and likewise supports use of an encrypted channel for additional security (HTTPS). Using XML and HTTP, Web services offer several benefits, including:

- Standardized, flexible application integration mechanism
- Avoidance of vendor lock-in (Web services are an open standard)
- Low cost of entry (simple technology that enjoys wide support)
- Built in, standardized mechanism to describe Web services using WSDL.
- Reside on top of web servers (IIS, Apache), gaining caching, security, session management, and scalability features.

2.1.4. Environmental Information Exchange Network

2.1.4.1. Exchange Network origin and participation

Driven by a growing hodgepodge of mechanisms to exchange environmental data, and a desire to build national, cohesive, and coherent environmental information systems, EPA and states formed the State/EPA Information Management Workgroup (IMWG) in 1998. The Environmental Information Exchange Network (Exchange Network) emerged from the work of the IMWG, as a unified mechanism to improve the exchange of environmental data between EPA, states, and other parties, using XML, Web services, and the Internet. Implemented in 2003, the Exchange Network is intended to enable better environmental decision-making by:

- Harnessing economies of scale through shared infrastructure and tools, thereby reducing costs

- Increasing data usage and integration among exchange partners
- Improve data quality through standardized, efficient data validation, while emphasizing early error detection
- Improved data availability and timeliness through automation
- Fostering new exchanges among states, EPA, and other partners

To support a wide variety of backend computing infrastructures in use by states, the Exchange Network is fully standards-based, with XML and Web services as foundation technologies. Since the implementation of the Exchange Network in 2003, states have gradually increased their participation in the Exchange Network (participation is presently voluntary). To encourage participation in the Exchange Network, EPA has provided over $98 million in grants to states and other environmental agencies, which has been instrumental in the success of the Exchange Network. Many states have also invested their own funds to participate in the Exchange Network.

2.1.4.2. Exchange Network data exchanges

The Exchange Network supports not only traditional data exchanges from states to EPA, but also between states, within states, and from EPA to states.

Table 4. Types of Exchange Network data exchanges

Exchange Type	Examples
State to EPA	Air emissions data to NEI Air quality data to AQS Hazardous waste data to RCRAInfo Drinking water data to SDWIS Facility data to FRS
EPA to State	Toxics data submissions Substance and chemical data
State to State	Common airshed data Common watershed data
Intrastate	Local government to state Drinking water labs to state

Data exchanges are also known as *dataflows,* representing a particular type of environmental data. The Exchange Network presently supports 19 production dataflows, and 13 additional dataflows are under development. In addition to the Exchange Network dataflows currently in development, several potential future dataflows are under consideration. The process for creation of a new Exchange Network dataflow involves formation of a Flow Development Group, which creates a XML Schema, Data Exchange Template, Flow Configuration Document, and several other documents. When complete, the documentation package is submitted to the Network Technology Group, which ensures conformance with Exchange Network design rules and conventions.

All Exchange Network dataflows occur with XML, which is validated according to the particular XML Schema that applies to the dataflow. All dataflows must include both an XML *payload* and *header*. The payload represents the source data that is specific to the particular dataflow. The header is generic, identifying the dataflow type, sender, contact information, submittal comments, and other supplemental information.

2.1.4.3. Pre-Exchange Network data exchange mechanisms

In the absence of the Exchange Network, many disparate environmental data exchange mechanisms evolved nationwide (Environmental Information Exchange Network, 2002). Due to this disparity and lack of standardization, participating in data exchanges was often costly, inefficient, and involved custom development of new transfer mechanisms for each type of data exchange. Moreover, transfers often involved manual processing and redundant data entry, resulting in inaccuracies and outdated data. Tight coupling of backend systems to transfer mechanisms further hampered standardized data exchange.

Table 5. Sampling of historical (non-Exchange Network) state to EPA data exchanges

Data Type	Transfer Mechanism	Transfer Destination
NEI	Flat text files or MS Access database	Manual upload to CDX Web
STORET	Oracle export file	Email or FTP to EPA

AFS	Flat text files	Upload to EPA mainframe using terminal emulator
SDWIS	Flat text files	Manual upload to CDX Web

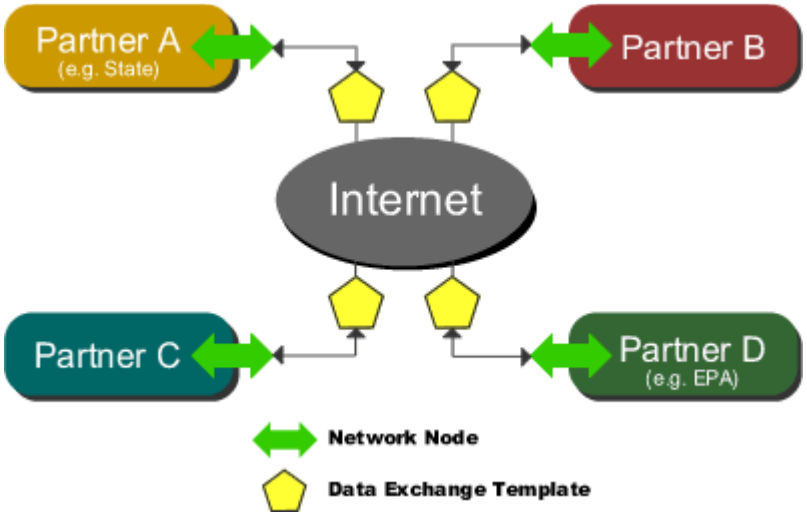
Historical, non-Exchange Network transfer mechanisms necessarily required custom validation programs to verify the format and content of environmental data. To provide data quality assurance, EPA would often provide states with custom executable programs to pre-validate submittals, and would periodically update these programs, as well as provide support for them. Yet given the high cost of creating and maintaining a wide array of proprietary validation programs (each type of environmental data required a separate, proprietary program), EPA would sometimes simply accept any submissions and validate it internally using database scripts. As both the number of environmental exchanges and data volume increased, creating and maintaining such validation programs became burdensome and inefficient, for both EPA and states. The Exchange Network provides an elegant solution, with a universal and standardized method to validate all types of data submittals using XML Schemas and Schematron. Moreover, submitters can validate their own XML data using the appropriate XML Schema, or utilize EPA's CDX validation Web service.

2.1.4.4. Exchange Network connectivity

States connect to the Exchange Network by first establishing a Exchange Network Node (Node), which represents the state's main point of presence on the Exchange Network. This Node is a web server exposed to the Internet, providing an encrypted channel using SSL. Using Web

services that conform to Exchange Network specifications, the Node listens and responds to data requests from other Nodes, and likewise submits data requests to other Nodes. In addition to providing a public interface, the Node is connected to various internal state databases to retrieve data needed for various dataflows. Each dataflow supported by the Node must be custom developed to "plug in" to the state's specific storage mechanism for the type of data in that dataflow.

Figure 2. Basic Exchange Network diagram



When a Node responds to data requests, the necessary data typically resides in a state relational database. Complex queries are often necessary to extract appropriate data from these databases, and must be converted to XML, which can involve significant overhead and result in poor response time. To improve performance, staging databases are sometimes used, where data has

been “pre-transformed” and “pre-joined.” An XML database (such as Software AG’s Tamino XML Server) is sometimes used for this purpose.

Since Exchange Network communications are based on standard Web services, any software program capable of calling Web services can communicate with a Node. Such programs are known as *Node Clients*. While a Node Client can request data from a Node or submit data to a Node, it cannot listen and publish data like a Node can, however.

2.1.4.5. Exchange Network security

Security is an important consideration in the Exchange Network. Traffic to all Nodes is encrypted using SSL, providing secure transport. For the purposes of authentication and authorization, the Exchange Network provides a central Network Authentication and Authorization Service (NAAS), which maintains information about all Exchange Network users and the privileges associated with each account. Prior to communication with any Node, a NAAS account ID and password must be passed to the Node, which in turn passes this account ID and password to NAAS for authentication. Use of NAAS alleviates the need to create custom authentication schemes, and enables single sign-on for the Exchange Network. Upon successful authentication, NAAS issues a token, which is passed to the Node for all action invocations during the session; this token expires after 10 minutes of inactivity.

2.1.4.6. Exchange Network Node 2.0

The specification for the next generation of the Exchange Network is presently under development, entitled Node 2.0. Primarily driven by vendor support issues and keeping the Exchange Network up-to-date with Web services standards, Node 2.0 will implement the following underlying technology changes:

- Message attachments will use Message Transmission Optimization Mechanism (MTOM) in place of Direct Internet Message Encapsulation (DIME).
- The SOAP 1.2 protocol will be used in place of SOAP 1.1.
- WSDL will use *document/literal* in place of *RPC-encoded* to define available services.

An implementation of Node 2.0 is expected to occur in 2008, with support for the present Node specification continuing through at least 2010.

2.1.5. National Emission Inventory

One of the dataflows supported by the Exchange Network is the National Emission Inventory (NEI), a conduit to EPA's massive NEI database with air pollution data from all 50 states, the District of Columbia, Puerto Rico, and the Virgin Islands. Primary (a.k.a. criteria) pollutants tracked in the NEI include carbon monoxide (CO), nitrogen oxides (NO_X), sulfur oxides (SO_X), volatile organic compounds (VOC), lead (Pb), ammonia (NH_3), and particulate matter (PM). The data in the NEI database is used by various entities for air dispersion modeling, tracking emission trends, regional strategy development, and regulatory decision-making. Table 6 notes the five types of NEI emission sources.

Table 6. NEI source types

Source Type	Description
Point	Specific air emissions from large facilities, such as power plants and oil refineries. Emissions are reported for each stack at the facility. Point source data comprises most of the data in a typical full NEI submittal.
Area	Aggregate air emissions from small individual sources, such as residential heating/cooling systems, lawnmowers, and fireplaces.
Onroad Mobile	Aggregate air emissions from automobiles.
Nonroad Mobile	Aggregate air emissions from motorized off-road mobile sources, such as aircraft, boats, and trains.
Biogenic	Aggregate air emissions from natural sources, such as forests, volcanoes, and wildfires.

States must submit NEI data for high emitting ("type A") point sources to EPA annually. NEI data for low emitting point sources ("type B") and all other non-point source types must be submitted every three years. A matrix in the CERR defines the various pollutant thresholds that distinguish between type A and type B point sources. Presently, there is a 17-month window for states to submit NEI data to EPA. For example, NEI submittals that covered the calendar year 2005 were due by May 31, 2007. EPA further has an 18-month period in which to process state NEI data and publish finalized, aggregate NEI data for public use to its web site at http://www.epa.gov/ttn/chief/eiinformation.html. This nearly 3 year delay in publishing NEI data is being accelerated over the next few years, as stipulated in the CERR.

NEI data can presently be submitted to EPA in three forms: text files, Microsoft Access MDB, or XML. The NIF specifies the required format for text files and Microsoft Access. EPA's Basic Format and Content Checker (BFCC) utility is a proprietary Visual Basic 6 application that can be used to validate text file and Microsoft Access NEI datasets. NEI XML files must be submitted to the CDX Exchange Network Node, and are validated using the NEI XML Schema and Schematron. With the implementation of the Exchange Network, EPA is encouraging a transition from the NIF to XML for NEI data submittals.

2.1.6. DEC AirTools application and source database

DEC's Oracle AirTools database serves as a data store for the AirTools application, a custom-built C# Windows Forms application used by DEC Air Division personnel for air permit tracking, compliance, emission inventory, and various other functions. The AirTools database contains a broad array of data, including all NEI source data. To populate NEI data in the AirTools database, DEC requests point source data from industry six to eight months prior to the NEI submittal deadline, receiving this data most often as paper and spreadsheets. Non point source data (area, non-road mobile, on-road mobile, and biogenic) are obtained through internal data modeling and data acquisition from various third parties. Both point and non-point NEI data are manually entered through the AirTools application by DEC data entry personnel, and stored in the AirTools database. This process occurs every year, where NEI submittals for large point sources must occur annually, and all others must occur every three years.

Running over the Red Hat Enterprise Linux ES 4 operating system, the Oracle 10g AirTools database consists of 156 tables, where 29 of these tables are pertinent to NEI data. Seven of the 29 tables related to NEI are core data tables, whereas the other 22 tables are secondary lookup and utility tables (see Appendix 1).

The AirTools database is accessible using Oracle SQL*Net, and ADO.NET, but only from within the state wide area network (WAN). NEXT will reside on a workstation within the state WAN with the Oracle client software installed, and will thus have direct access to the AirTools database. A specific NEI Oracle account has been created for the purposes of this project, which provides read only (i.e. SQL SELECT) access to the emission inventory tables. Appropriate permissions have also been granted to create needed stored procedures under this NEI Oracle account.

The AirTools database generally adheres to relational database best practice design principles. Numeric primary keys are defined for all tables, with appropriate foreign keys to ensure referential integrity. To ensure data integrity, all tables are organized into third normal form (3NF). Several table indexes have been created to enhance query performance, based on reviews of AirTools usage by DEC's Oracle database administrator (DBA). Moreover, all tables and columns in the AirTools database include textual comments describing the data they store.

The DEC data center houses both a production and development Oracle database, on different physical servers. In support of ongoing development work, AirTools tables and data are periodically replicated from the production database to the development database, as needed. A production to development replication for all 2005 NEI data was recently performed by the Oracle DBA, ensuring that a real world set of data can be used for development work on this project. To ensure no adverse impact on the production AirTools database, this project will exclusively reference the development AirTools database.

2.1.7. DEC existing NEI export program

Since 1999, DEC's existing custom-built Alaska Emission Inventory System (AEIS) software has been used to extract AirTools NEI data and transform it into the NIF text format. AEIS was written using Borland Delphi 5, and DEC has retained all the source code. AEIS serves as the first step in the existing NEI extract, transform, validate, and submit process:

1. Extract NEI data from the AirTools database and transform it into the NIF text format using DEC's AEIS.
2. Validate resulting NIF text file format and content using EPA's BFCC utility. If errors are reported, correct the appropriate AirTools database source data, and start over at step 1. The BFCC cannot validate biogenic source type NIF files.
3. Name NEI data file according to CDX requirements, and compress using ZIP format.
4. Logon to CDX Web, fill in NEI submittal information, and upload validated and compressed NEI data file.

This process is repeated for each of the five different NEI source types.

In terms of technical architecture, AEIS is a client/server Windows application, and uses native Oracle drivers to improve performance (Allround Automations' Direct Oracle Access product). Despite the highly data-intensive nature of this application, the AEIS database interface is inefficient, consisting exclusively of client-side SQL calls (no stored procedures are used). Moreover, SQL statements are dynamically constructed in the application, and lack bind variables. This approach requires hundreds of SQL calls that cannot utilize Oracle's statement cache, resulting in runtimes in excess of 15 minutes for an NEI point source export for a typical year. While a 15-minute run time is not problematic for a single run, it becomes highly problematic when several runs need to be performed as validation errors are discovered and corrected. To ensure usability and efficiency, NEXT should extract and transform all NEI point source data for a given calendar year to XML within three minutes. Upon successful implementation of NEXT into the production environment, AEIS will be retired.

2.1.8. EPA NIF to XML converter

To assist with the transition to use of XML, EPA provides a free executable utility that is intended to convert NEI NIF text or Microsoft Access files to NEI XML. Written for the .NET Framework 1.1, this utility is presently in beta, and was last updated in September 2005. This converter is not considered as a viable alternative to this project, because it has no programmatic interface, and cannot submit NEI data to EPA. Nonetheless, the software was tested for its potential value as a comparative product. An initial evaluation uncovered several problems, most seriously when the software was unable to process two sample Alaska NIF text files, aborting with an unhandled (and non-descript) exception. Upon contacting EPA regarding these

problems, the author discovered that no source code is available for the utility, and EPA has suspended its development due to its general inapplicability to state NEI Exchange Network integration efforts. As a result, this utility was not used for this project beyond the initial evaluation.

2.2. The contribution this project will make to the field

This project will comprise a full software development project, from requirements analysis to implementation, and can serve to inform and enlighten others in several respects. In general IT terms, this project will represent a useful case study about the viability, benefits, and problems when developing an adapter to allow an existing application to participate in an SOA. Moreover, moving data efficiently between disparate systems (both internal and external) is a significant concern in many organizations, and this project can provide valuable insights when considering XML for this purpose. This project is also of specific interest to those organizations planning or considering utilizing the Exchange Network. As the first Exchange Network dataflow at DEC, this project will be of particular interest as DEC determines whether those additional dataflows will be pursued.

2.3. Summary

Generic, core technologies pertinent to this project include XML and Web services, which are open technologies used for data integration. As an SOA, the Environmental Information Exchange Network is a unified mechanism to improve the exchange of environmental data

between EPA, states, and other parties, using XML, Web services, and the Internet. The Exchange Network supports various dataflows (different types of environmental data), one of which is the National Emission Inventory (NEI). DEC has a legacy NEI transfer program that generates text files, but does not validate this data or submit it to EPA. The existing AirTools database will provide all needed NEI data for NEXT. This project will provide a valuable case study for participating in an SOA, as well as using XML for data exchange.

3. Chapter Three: Methodology / Plan

3.1. Life cycle model to be followed

The software development life cycle to be generally followed on this project is the Sashimi Model (aka Waterfall with Overlapping Phases Model). The pure Waterfall Model consists of requirements analysis, design, construction, testing, and implementation phases, where these phases are disjoint and completed sequentially -- the work in one phase is fully completed before starting the next phase. The Sashimi Model allows for overlap between adjacent phases, which allows more flexibility as the project progresses (McConnell, 1996). While the goal and requirements of this project are clear, some flexibility in project phases is desirable to address emergent issues as design and development work proceeds.

During the requirements analysis phase, the problem domain will be investigated in depth, to include review of pertinent Exchange Network documentation, the existing AirTools database, XML, and corresponding XML Schema. Because the project feasibility, scope, and resource commitment have already been established, these aspects will not be considered during requirements analysis. The design phase will address application architecture, integration into the existing DEC computing infrastructure, database interfaces, user interfaces, and error handling. The software will be built in the construction phase, followed by the testing and implementation phases. The Waterfall Model also traditionally includes a post-delivery maintenance phase. However, since the project does not involve deployment to the production server, the maintenance phase is excluded (a subsequent production deployment will naturally have a maintenance phase, however).

While alternative life cycle models such as Rapid Application Development (RAD), Extreme Programming (XP), Agile, and Spiral were considered, the Sashimi Model was ultimately selected as the optimal life cycle on a project of this nature, for the following reasons:

- The desired application functionality is unambiguous.
- The application scope is fairly small.
- The application requirements are largely defined in the Exchange Network NEI documentation, in particular the NEI Flow Configuration Document, NEI Data Exchange Template and NEI XML Schema.
- The Sashimi Model offers a sequential phase progression, and allows for some overlap between adjacent phases, which provides desirable flexibility.

3.2. Specific procedures

3.2.1. AEIS evaluation

The project will begin with an evaluation of the existing AEIS program and transfer code. Built in 1999, AEIS transforms data from the AirTools database into a fixed length text format that conforms to the NEI NIF specification (see section 2.1.5). No system documentation exists for AEIS, so this evaluation will consist of a review of Borland Delphi source code and forms. A key goal of this evaluation is to identify appropriate columns, indexes, and queries that comprise NEI data. In conjunction with this effort, EPA's document that details mappings between NEI NIF data elements and the corresponding NEI XML Schema elements will be referenced.

3.2.2. AirTools database

The existing AirTools Oracle 10g relational database will be referenced by NEXT as the sole source repository for all necessary NEI data. Since the AirTools database is used by the existing AirTools application (see 2.1.6), this database will not be modified in any way as part of this project. Pertinent documentation of the AirTools database will be extracted using Microsoft Visio, which can create an Entity Relationship Diagram (ERD) using its reverse engineering capability. NEI related tables and columns are also fully commented using Oracle's object comment capability. If Visio cannot extract these comments, they will be extracted using appropriate queries against the Oracle data dictionary.

3.2.3. NEI data elements

All NEI data elements are characterized as either *mandatory*, *necessary*, or *optional*. Mandatory data elements are required; an NEI submittal without these data elements will be rejected by EPA. Necessary elements are desirable, but EPA will plug in modeled values for these data elements if they are not provided. Optional data elements are fully optional. This project will only address mandatory and necessary data elements.

3.2.4. Development tools

A Microsoft Visual Studio 2005 C# Windows Forms project will be used to build the NEXT front-end. EPA provides a .NET Client Toolkit, which provides example C# code for various Exchange Network Node Web service calls. This toolkit will be used an important point of

reference in the development of NEXT, as well as a source of code reuse. Allround Automations' PL/SQL Developer will be used for creation of PL/SQL back-end Oracle stored procedures.

3.2.5. Source control

An existing DEC Microsoft Visual Source Safe 2005 (VSS) source control system will be used for the project. Although only a single developer will be coding for the project (that is, the author), use of VSS will provide full versioning, history, and backup of all source code changes. The actual VSS repository is stored on a DEC file server, which is also available via a VPN for remote development work. The Visual Studio 2005 C# project will connect using built-in VSS plug-in capability. The PL/SQL Developer project will use a third party plug-in to reference the same VSS repository.

3.2.6. Software engineering paradigm

The author is a strong proponent of object-oriented analysis, design, and programming, as an object-oriented approach can offer a host of benefits (Schach, 2005). However, the nature of this particular application is not well suited to full object-orientation. A key function of NEXT is extraction of a large set of data elements from a relational database and transformation to XML. Given stringent performance requirements (see 2.1.7), NEXT must incur minimal overhead, and the process of reading from the database, transforming, and writing to XML must be direct and efficient. Inserting an intermediary layer of classes/objects between the database and XML

would not only result in performance degradation, but increased application complexity. Moreover, significant NEXT processing will occur within PL/SQL stored procedures, which negates the ability to represent business rules there as operations within objects. As such, the extract and transform functionality in NEXT will generally be designed and built using a classical, rather than object-oriented paradigm. The validate and submit functionality in NEXT will be designed and built using an object-oriented paradigm, however.

3.2.7. Application architecture

In accordance with software design best practices, NEXT will be logically partitioned into a structural framework, consisting of the following logical layers (Fowler, 2003):

- Presentation— user interface (implementation is Windows Forms)
- Domain— objects that describe the problem domain (excludes objects that directly map to NEI database tables)
- Business— business rules (in addition to internally defined rules, will reference business rules defined in PL/SQL)
- Integration— database access (implementation is Oracle RDBMS)

Layers will be implemented as distinct and appropriately named folders within the NEXT Visual Studio Project. NEXT will be physically partitioned onto two tiers: client (workstation) and server (database).

NEXT layers and application components within layers will adhere to the following general best practice design principles (IEEE Computer Society Professional Practices Committee, 2004):

- Encapsulation— Entity elements and internal details are packaged, such that those details are hidden.
- High cohesion— Entity elements and internal details are all strongly related, contributing to a single purpose.
- Low coupling— An entity has few dependencies on other entities.
- Modularity— Large entities are compartmentalized and decomposed into smaller independent entities.

Wherever possible, appropriate architectural and design patterns will be implemented in NEXT, to utilize proven, tested designs for common object-oriented design problems. In particular, the following architectural and design patterns will be considered (Fowler, 2003):

- Separated Interface— Defines an interface separately from its implementation.
- Abstract Factory— Provides an interface for creating related objects without specifying their concrete classes.
- Plugin— Links classes during configuration rather than compilation.
- Façade— Provides a simplified interface to a larger, complex body of code.

Since NEXT will not implement an object layer as an intermediary between the source data and XML (see 3.2.6), datasource-related architectural patterns will not be used in the integration layer (such as Table Data Gateway, Row Data Gateway, Active Record, and Data Mapper).

3.2.8. Coding comments

The author will be responsible for writing all code for the project, including both C# and PL/SQL. To ensure clarity upon subsequent reviews, all code will be thoroughly and completely documented using in-code comments. In C#, each class, method, and property will be documented using the XML comments feature (invoked using a triple forward slash). In PL/SQL, each procedure will be commented immediately above the procedure header. All comments will include a general description of the item, along with parameter descriptions and data types. Additional comments within methods or procedures will be liberally added, as necessary.

3.2.9. Testing

Appropriate NEXT unit tests will be created using the NUnit open source unit-testing framework. Unit tests will be created that have a reasonable probability of catching an error, based on the judgment of the author (Kaner, Falk, & Nguyen, 1999). While the CDX Validation Web service provides an official validation assessment, unit tests can provide further assurances that NEI data is being properly extracted from the AirTools database. All errors discovered during informal and unit testing will be logged into DEC's web based bug tracking system.

Best practices indicate that software development and testing should be conducted by different individuals or groups (Kaner, Falk, & Nguyen, 1999). Beta testing by end users is often desirable as well. However, individuals other than the author will probably not be enlisted to

assist with testing on this project, due to the nature of the application. NEXT is a utility program with a very specific purpose and limited user interface (likely consisting of simple Exchange header data entry and a "Go" button). Given this, the most useful and meaningful tests for NEXT are automated tests, particularly unit tests and calls to the CDX validation Web service.

3.3. Formats for presenting results/deliverables

The C# project application will be provided as a Microsoft Visual Studio 2005 solution, consisting of all appropriate solution folders and full source code. PL/SQL will be provided in a SQL script that will create the PL/SQL within a self-contained Oracle package. System documentation and logs will be provided as Microsoft Word files. Performance testing results will be provided in a Microsoft Excel spreadsheet.

3.4. Review of deliverables

As noted in Table 7, this project will culminate with six specific deliverables.

Table 7. Project deliverables

Number	Item
1	Microsoft Visual Studio 2005 C# Windows Forms solution for the NEXT front end (see Appendix 2 for application screenshots). NEXT must extract NEI data from the AirTools database, transform this data into XML, validate the XML, and finally submit the XML to the CDX Test Node. NEXT must process all five NEI data

	source types (point, area, onroad mobile, nonroad mobile, and biogenic). NEXT must also consider non-functional quality factors, including reliability, usability, maintainability, extensibility, and adaptability.
2	Oracle PL/SQL script to create stored procedures utilized by deliverable #1.
3	Application documentation (requirements specification, ERD, class diagram, and sequence diagrams)
4	Log of a failed (invalid) NEI submittal to the CDX Test Node. XML validation errors should be explicitly shown in the log (see Appendix 4).
5	Log of a successful NEI submittal to the CDX Test Node for all Alaska's 2005 air data, to include all five source types (point, area, onroad mobile, nonroad mobile, and biogenic).
6	Performance testing results for Alaska 2005 point source NEI data, to include extract (query) time, XML write time, compress time, validation time, and submit time. The extract + XML write time must not exceed three minutes, and tests will be conducted on the author's development laptop.

The project schedule noted in Table 8 spans a 77-day period, from August 6 to October 21. Although the final task is scheduled to be completed by 10/14, the project completion deadline is officially October 21, which leaves seven days of slack to cover potential task extensions.

Table 8. Planned project schedule

Task ID	Task Description	Estimated Work Effort (hours)	Start Date	End Date
1	Exchange Network Research and Requirements Analysis	25	8/6	8/26
2	Application design	15	8/27	9/9
3	Application development	90	9/10	10/7
4	Application testing & submit data to CDX Test Node	15	10/8	10/14

3.5. Resource requirements

Hardware requirements include a Dell Dimension D620 Laptop (2GB RAM, Intel T7400 CPU) and a Dell PowerEdge Server (4GB RAM, Dual Intel Xeon CPUs, RAID5 drive array). Software requirements include Oracle 10g relational database, Microsoft Visual Studio 2005, Altova XMLSpy, Microsoft Windows XP, Microsoft Visio, Microsoft Word, and Microsoft Excel. The development Oracle 10g database resides on the Dell PowerEdge Server (the production Oracle database will not be used for this project). All the required hardware and software is already available, having been purchased by DEC for use on other projects. Personnel resource requirements include the author, along with potential limited assistance from other DEC personnel and EPA CDX support personnel.

3.6. Outcomes

The principal functional outcome of this project is custom software capable of successfully extracting, transforming, validating, and submitting Alaska NEI data to the Exchange Network CDX Node. Supplemental nonfunctional outcomes include suitable performance (maximum of three minutes to extract and write point source one year of NEI data), and completion prior to the project deadline (October 21). Collectively, these outcomes comprise the project success criteria. If successful, this project will form the basis for development of further dataflows at DEC. If not successful, an assessment of failures and potential remedies may occur, or DEC initiatives in support of the Exchange Network may be withdrawn entirely.

3.7. Summary

This project will utilize the Sashimi Life Cycle model, as an appropriate approach based on project characteristics, while providing increased flexibility from the pure Waterfall Model. The project will include an evaluation of the existing NEI transfer mechanism, namely AEIS, the AirTools database, and NEI data elements. Microsoft Visual Studio will be used to develop the Windows Forms NEXT front end, and PL/SQL Developer will be used to develop back end Oracle PL/SQL. Best practice software engineering principles will be implemented when developing NEXT, including layering and integrated unit testing. To ensure optimal application performance, NEXT will be developed using a hybrid object-oriented and classical paradigm. At the end of a 77-day project work schedule, NEXT will be delivered as a software product capable of submitting Alaska NEI data to the CDX Test Node.

4. Chapter Four: Project History

4.1. How the project was managed

This project was generally managed using Traditional Project Management (TPM) techniques (Wysocki & McGary, 2003). The author wore many hats on this project, including project manager, developer, tester, and technical writer. Much of the basis for managing this project was derived from the project objectives and deliverables. With a clear purpose and unambiguous deliverables, the scope of the project was well defined. Known project risks (described in section 1.4) were also clearly identified at the outset of the project, and continuously monitored throughout the project. Fortunately, no risks related to format or regulatory changes manifested themselves during the project, which allowed the project to be completed within schedule.

The project plan consisted of identification of project activities, timelines, and resources requirements. While project activities were clearly defined, activity work effort was difficult to determine, because a project of this nature had not been performed at DEC before. An educated guess was essentially provided, based on the experience of the author, review of EPA's example C# .NET Client Toolkit, and brief discussions with other states agencies that have participated in the Exchange Network. The project schedule made a distinction between task work effort and duration, which provided a more accurate assessment of task work, given that the author had other non-project related responsibilities during this time.

To track specific coding tasks and desired fixes, TODO tags were used in both Visual Studio and PL/SQL Developer. Use of these tags provided a handy way to track coding changes and work

tasks remaining. With several thousand resulting lines of PL/SQL and C# project code (excluding auto-generated code, comments and blanks), this capability proved to be essential. Visual Source Safe was also used to track aggregate source code changes, and the author referenced prior code versions in VSS on several occasions.

To close out the project, documentation was finalized, and the project deliverables were created and stored on a designated location on the DEC file server. A project summary report was also created and distributed to various DEC IT personnel with potential interest in use of the Exchange Network.

4.2. Significant project milestones/events

The actual project progression generally followed the scheduled plan, but deviated on task #3, which was intended to be completed by 10/7, but was not completed until 10/15, due to various technical difficulties (see section 4.5). The actual work effort was also correspondingly higher for task #3, being 112 hours instead of the planned 90 hours. Nonetheless, as noted in Table 9, the project was still completed prior to the 10/21 deadline, due to the inclusion of slack time in the original project schedule.

Table 9. Actual project schedule / milestones

Task ID	Task Description	Actual Work Effort (hours)	Completed Date
1	Exchange Network Research and Requirements Analysis	19	8/26
2	Application design	18	9/9
3	Application development	112	10/15
4	Application testing & submit data to CDX Test Node	16	10/20

4.3. Changes to the project plan

The primary change to the project plan was the extension of the task #3 completion date by eight days, due to various difficulties encountered during development. Yet, even with this change, slack time in the project schedule allowed completion by the October 21 deadline.

Project deliverable #5 required the demonstration of successful data submittals for all NEI source types to the CDX Test Node, based on 2005 NEI data in the AirTools database. Prior to any submit operation, a successful validate operation must first occur. However, initial validation runs indicated some 250 problems with source data, which were mostly either "out of range" or "invalid code." To correct this and proceed to submit operations, the author manually adjusted all reported errors to known valid values using the AirTools application front end, along with

direct AirTools database SQL update statements. This data correction effort consumed a significant amount of time, and was not part of the project plan. While this possibility was considered as a risk factor (see 1.4.4), the effort involved was not quantifiable prior to the project outset.

4.4. Evaluation of whether or not the project met project goals

The project met its primary goal of developing software with a demonstrated ability to successfully submit Alaska NEI data to the CDX Test Node, including all five NEI source types. Secondary project goals were met as well, including project completion by October 21, ensuring the extract and XML writing process for NEI point source data occurred within three minutes, and satisfaction of all six project deliverables.

4.5. What went right

In retrospect, several aspects of the project contributed to its success. In particular, a clear and unambiguous project goal, clearly and fully documented project requirements, and an organized management approach kept the project on track and steadily moving toward the project goal.

The use of programming languages (PL/SQL and C#) that were well known by the author was a key decision. The project timeframe was aggressive, and if languages unfamiliar to the author were used, the project could not have been completed within the allotted timeframe. Moreover,

use of a comprehensive suite of NUnit unit tests proved to be valuable as a quality assurance tool as development progressed and various coding changes were implemented (see Appendix 6).

A significant project timesaver proved to be use of EPA's CDX validation Web service, instead of attempting to validate XML locally. While validating remotely is distinctly slower than locally, use of the CDX validation Web service simplified the project by alleviating the need to write validation code. Additionally, while the .NET Base Class Library includes XML Schema validation capabilities, it does not include Schematron validation capabilities (although several open source Schematron validation libraries for .NET do exist).

Use of PL/SQL as the workhorse in extracting NEI data was a further key decision, as the broad scope of data elements and processing involved became clear. Moreover, by aliasing database columns to appropriate XML data element names in PL/SQL, the C# portion of NEXT could focus exclusively on writing out the XML and submitting it to the CDX Test Node.

4.6. What went wrong

No aspects of NEXT development would be appropriately classified as something that "went wrong." However, while the overall data transformation process and Web service exchanges appeared deceptively straightorward at the project outset, several difficulties emerged as development work delved deeper into specific application details. These difficulties were both of an administrative and technical nature, as noted in Table 10.

Table 10. Project difficulties

Problematic Aspect	Description
Node Help Desk support	EPA's contractor-run Node Help Desk is the sole provider of Exchange Network technical assistance. The author attempted to obtain assistance from the Node Help Desk on several occasions. Unfortunately, obtaining meaningful help from the Node Help Desk was often a slow, multi-day process.
Regulatory and data format inconsistencies	The CERR is the federal rule that mandates submittal of periodic emission inventory data to EPA. The CERR lists specific data element requirements, as does the NEI XML Schema. However, the data requirements of the CERR and NEI XML Schema are sometimes inconsistent. In a discussion with EPA regarding this discrepancy, EPA confirmed that they *should be* the same. EPA further indicated that either the CERR or NEI XML Schema *should be* updated to match in the near future. Despite this unresolved discrepancy, NEXT will conform to the NEI XML Schema, as it must in order to create valid NEI submittals.
Web service attachment mechanism	The Exchange Network requires transmittal of source data as compressed (zipped) XML data files. When invoking a Web service submit call, this zipped data is attached to the SOAP call itself, using DIME (by attaching binary data outside the SOAP body, costly encoding and decoding is avoided). However, the BCL does not include support for DIME by default-- download and installation of Microsoft's Web Service Extensions (WSE) was required. Only after considerable troubleshooting

	did the author discover that an automatically-generated C# Web service reference file must be manually altered to use DIME (reference.cs). DIME has since been superseded by MTOM, which is evidently easier to use and more efficient than DIME. The Node 2.0 specification calls for use of MTOM.
File compression format	Prior to initiating an Exchange Network submit Web service call, the source XML file must be compressed using the ZIP format. While the .NET BCL includes compression functions, the GZIP format used in the BCL is incompatible with ZIP format required by the Exchange Network. As such, an open source compression library that supports the ZIP format was utilized instead (#ziplib).
Faulty XML validation criteria	In one instance, the validation provided at the CDX validation Web service was incorrect. Initial submittals to this service were rejected due to stack heights that exceeded 100 feet (yet real stack heights often exceed 100 feet). After informing the Node Help Desk of this problem, they altered the NEI Schematron validation to permit stack heights up to 1,000 feet.
Submit permissions problem	Although the author had an established NAAS account that should allow submission to the CDX Test Node, a permissions error was received the first time the author attempted to submit NEI data. In contacting the Node Help Desk, a specific submit permission for NEI data had improperly not been added to the account.

Limited Oracle column name length	According to the application design, the extracted Oracle column names would be aliased with appropriate XML element names. For example, column name `FCT_EMISSION_PERIODS.START_DATE` was aliased to `EmissionPeriodStartDate`. This approach generally worked well. However, Oracle has a 30-character limit on column names, and some required NEI XML element names exceeded 30 characters. This situation necessitated the need to create shorthand column name components (e.g. `Em -> Emission`), where the elongation then occurred in the C# client application just prior to writing the XML. While somewhat inelegant, Oracle's limits in this regard do not leave any other real alternatives.
Null data handling	When a particular source database column contains a null value, there are two different approaches to handling this when writing to XML. The initial default approach using C# was to write empty tags (e.g. `<EmissionPeriodStartDate>< /EmissionPeriodStartDate>`). However, files with empty elements coded in this manner were rejected by the CDX validation Web service. It turned out that the CDX validation Web service requires the alternative approach, which is to exclude the data element entirely.
Byte order mark	All initial XML submittals to the CDX Test Node were rejected for having an "invalid file format." After considerable troubleshooting, and

	using a hex editor, the author discovered that the BCL XmlTextWriter class by default writes a Byte Order Mark (BOM) in the first three bytes of the resulting XML file. The CDX validation Web service was confused by the presence of these three bytes. A specialized call to the XmlTextWriter constructor was necessary to omit the BOM.
Cryptic validation reports	CDX validation Web service validation reports tend to be fairly cryptic (see Appendix 4). When a validation error report is received, in order to correct problems, the submitter must map each error in the report to the appropriate data instance in the source application/database (e.g. AirTools). With this mapping, the source data can be corrected, and the extract performed again. However, such mappings can only realistically be performed by an individual with requisite knowledge of both the XML and source application/database (as was the case on this project).

4.7. Findings/Results

With the successful submission of Alaska NEI data to the CDX Test Node on October 20, NEXT achieved the primary project goal (see Appendix 3). Secondary project goals were met as well, including project completion by October 21, ensuring the extract and XML writing process for NEI point source data occurred within three minutes (the actual time was less than one minute, as illustrated in Appendix 5), and satisfaction of all six project deliverables. The success of the NEXT project represents a promising step towards standardizing DEC's environmental dataflows, and gaining the full benefits the Exchange Network offers.

The Exchange Network is a necessary and appropriate national solution to the problem of disparate environmental data transfer mechanisms. Developed over several years as a collaborative effort between states and EPA, the Exchange Network is a robust and carefully crafted means to improve environmental dataflows. Moreover, with a technological foundation based on XML, Web services, and the Internet, partners can universally participate in the Exchange Network, regardless of their own internal computing infrastructures.

NEXT provides significant improvements over AEIS in processing NEI data. While AEIS functionality was limited to extract and transform, NEXT provides a fully integrated solution, including extract, transform, compress, validate, and submit functionality. The Exchange Network facilitates integration of the validate and submit functions through Web services, while a NEXT design steeped in software engineering best practices facilitates integration and optimal processing for the extract, transform, and compress functions. As a result, the extract and transform process was reduced from over 15 minutes in AEIS to less than one minute in NEXT (this improvement is unrelated to the Exchange Network, per se). Although the raw XML NEI files generated from NEXT are considerably larger than the corresponding text files generated from AEIS, compressing XML files prior to submission reduced their size by over 90%.

While the development of NEXT was ultimately successful, it was more difficult than anticipated. Several technical difficulties hampered development (in particular tasks relating to Web service calls), resulting in a delayed project completion date. Some difficulties were likely exacerbated by the author's lack of experience in working with XML and Web services, and

lacking Exchange Network technical support. While the project was completed nonetheless, these difficulties do underscore short term Exchange Network cost of entry and learning curve considerations for new Exchange Network participants such as DEC. However, a key Exchange Network benefit is achieving economies of scale, which can only be realized over the long term as Exchange Network participation increases.

4.8. Summary

Using Traditional Project Management techniques, the author wore many hats on this project, including project manager, developer, tester, and technical writer. Well-documented requirements and a clear project goal kept the project focused. The project was successfully completed prior to the project deadline, although the development task was eight days longer than planned. Use of familiar programming languages, the CDX validation Web service, and heavy use of PL/SQL proved to be instrumental project success factors. Various technical difficulties were encountered as NEXT development progressed, although none was insurmountable. NEXT succeeded in its goal to submit NEI data to the CDX Test Node, and proved to be a significant improvement over AEIS.

5. Chapter Five: Lessons Learned and Future Project Evolution

5.1. Conclusions

Development of NEXT and a successful submission to the CDX Test Node demonstrated the viability of adapting an existing data exchange process to use XML, Web services, and the Exchange Network. Although NEXT is a specific, limited example, this generically demonstrates the viability of developing an SOA service requester (NEXT) that references local data and connects to an existing SOA service provider (CDX). Moreover, because an SOA is standards-based, most modern development tools can be used to develop requesters and providers (for canned enterprise software, adapters are often available as well). Although most SOA literature addresses implementations where requesters and providers exist within a single enterprise/organization, SOA is an appropriate architecture for environmental dataflows nationwide.

All hype and propaganda aside, the promise of SOA is true, offering real benefits (Erl, 2005). In an increasingly complex and heterogeneous IT environment, SOA provides a consistent architectural framework where applications can be rapidly developed, integrated, and reused. The SOA ideal is also achievable. In the development of NEXT, the existing AirTools database and application were unchanged, leveraging existing technology investments and abstracting these backend systems from the Exchange Network. The end-to-end automation achieved in NEXT was compelling, being realized by the ability to reference validation and submit Exchange Network services. The benefits of reuse and standardization were also plainly evident, as all states utilize Exchange Network services in the same manner. However, realization of full SOA

benefits will only occur as participation increases (Erl, 2005). DEC will further realize Exchange Network benefits with the implementation of additional Exchange Network dataflows.

Standardization enables SOA. For a Web services SOA implementation, technology standards include XML, SOAP, WSDL, and UDDI; the underlying transport most often uses TCP/IP. Widely accepted and open technology standards such as these allow efficient integration and abstraction of heterogeneous source data systems and tools, be it .NET, Java, Oracle, SQL Server, etc. SOA services are interoperable, being decoupled from backend technologies, platforms, and operating environments. In addition to technology standards, data standards are key to realizing SOA benefits, by laying the groundwork for data integration. In the Exchange Network, environmental data standards are defined using XML Schemas and Schematron, providing a consistent means to describe and validate all environmental data.

As with many new paradigms and technologies, the initial costs and learning curve for SOA participation can be high. Service oriented solutions not inherently simple or easy to build (Erl, 2005). While none was insurmountable, several technical difficulties occurred during NEXT development, delaying the project completion date. The initial effort and cost to participate in an SOA will vary depending on the nature of the application or business function intended for SOA integration, along with the skills and experience of designers and developers. In any case, a transition to SOA will demand effort, discipline, and time.

While SOA has tremendous potential benefits, caution must be exercised when considering SOA, and quality can vary. A poor quality or ill-conceived SOA implementation can result in detrimental software architectures (Erl, 2005). Use of services does not negate the need for software engineering best practices, understanding of underlying business rules and processes, or common sense. The interoperability gains achieved in an SOA implementation using Web services do introduce potential performance degradation due to the overhead associated with XML writing, XML parsing, and transmitting sizeable SOAP messages. For NEXT, the performance impact was negligible, largely due to ZIP compression of the NEI data XML attachment, a coarse-grained service interface (the Submit is a single Web service call), and an asynchronous call design (after the Submit call, processing status is retrieved via a GetStatus call). Appropriate preparations must also occur to lay the groundwork for SOA. In particular, a robust, optimized, and interoperable SOA can only be created by first standardizing the manner in which data is represented, validated, and processed.

5.2. Project Evolution / Recommendations

The successful development of NEXT and submission to the Exchange Network represented the first phase in an evolving feature set for this software. Five recommendations for furthering the base that was established in NEXT are noted in Table 11.

Table 11. Project recommendations

Synopsis	Description
Deploy to	NEXT should be deployed into production in support of the 2006 NEI data

production Oracle database server	submittal, which must occur prior to 5/31/2008. The PL/SQL script must be executed on the production Oracle server to create the appropriate packages. The C# application configuration file (app.config) must be altered to reference the production server.
Implement further DEC dataflows	NEI is the first Exchange Network dataflow implemented at DEC. Future potential dataflows include Safe Drinking Water Information System (SDWIS), Facility Registry System (FRS), and Water Quality and Discharge (ICIS-NPDES). Still further dataflows may be developed at DEC as support for new dataflows is added to the Exchange Network.
Automate NEI point source data retrieval	All point source NEI data is obtained directly from industry. When obtained, this data is manually entered into the DEC AirTools application, which correspondingly fills NEI related tables. This data collection effort represents a very large data entry burden for DEC each year. A complementary project would be to develop a system to facilitate automated submittals of point source NEI data directly from industry. Large companies may be well suited to provide NEI data directly in the XML format, whereas smaller companies might be able to utilize a web site to load this data.
Proactively address upcoming regulatory and technology changes	The function of NEXT is tied to a dynamic regulatory and technology environment. Several changes are forthcoming which will affect NEXT, in particular Node 2.0, NIF 4.0, and CERR. While none of these changes in their present form will require major NEXT modifications, the status of these potential changes should be closely monitored and considered as

	NEXT development proceeds.

5.3. Summary

The successful development of NEXT demonstrated the viability of participating in an SOA, by developing an adapter as a bridge from an existing system. An SOA offers real benefits, including the ability to preserve legacy systems, automation, and standardization. SOA quality can vary, and SOA pitfalls must be well understood prior to embarking on an SOA implementation. NEXT is recommended to evolve in several respects, including deployment to production, serving as a basis for further DEC dataflows, automating source data retrieval, integrating in the DEC Exchange Node, and addressing upcoming regulatory and technology changes.

References

Britton, C., & Bye, P. (2004). *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems* (2nd ed.). Boston: Unisys Corporation.

Environmental Information Exchange Network. (2006, August 3). Exchange Design Guidance and Best Practices for the Exchange Network.

Environmental Information Exchange Network. (2002, February 12). Implementation Plan for the National Environmental Information Exchange Network.

Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design.* Upper Saddle River: Pearson Education.

Feuerstein, S., & Pribyl, B. (1997). *Oracle PL/SQL Programming.* Sebastopol: O'Reilly.

Fowler, M. (2003). *Patterns of Enterprise Application Architecture.* Boston: Addison-Wesley.

IEEE Computer Society Professional Practices Committee. (2004). SWEBOK 2004 Version.

Kaner, C., Falk, J., & Nguyen, H. Q. (1999). *Testing Computer Software.* New York: Wiley.

MacDonald, M. (2003). *Microsoft .NET Distributed Applications: Integrating XML Web Services and .NET Remoting.* Redmond: Microsoft Press.

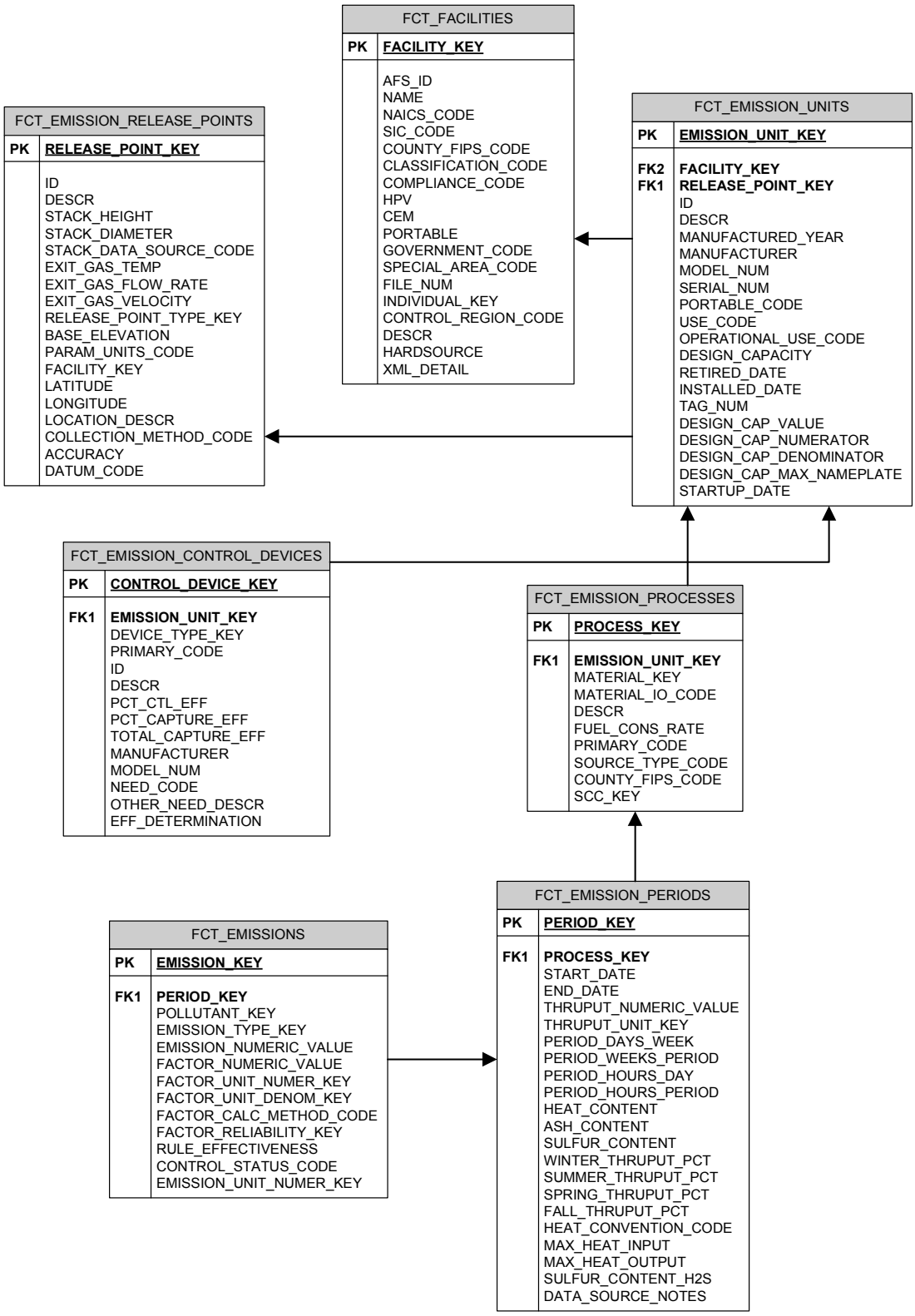
McConnell, S. (1996). *Rapid Development.* Redmond: Microsoft Press.

Ray, E. T. (2003). *Learning XML* (2nd ed.). Sebastopol: O'Reilly.

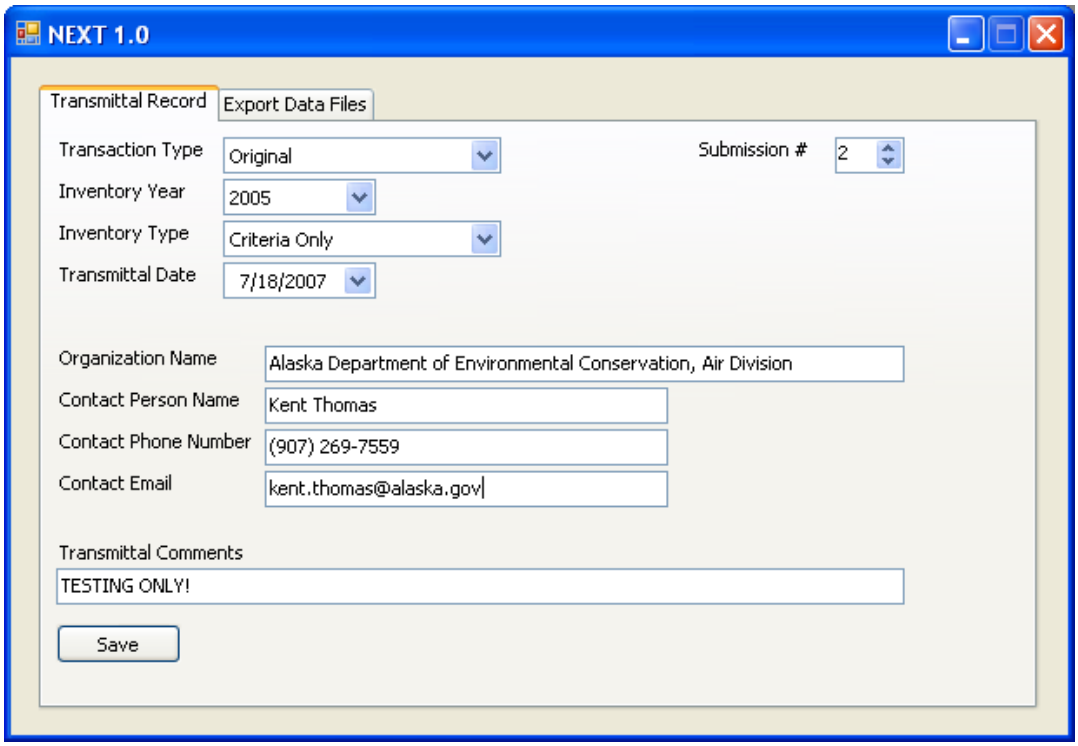
Schach, S. R. (2005). *Object Oriented and Classical Software Engineering* (6th ed.). New York: McGraw-Hill.

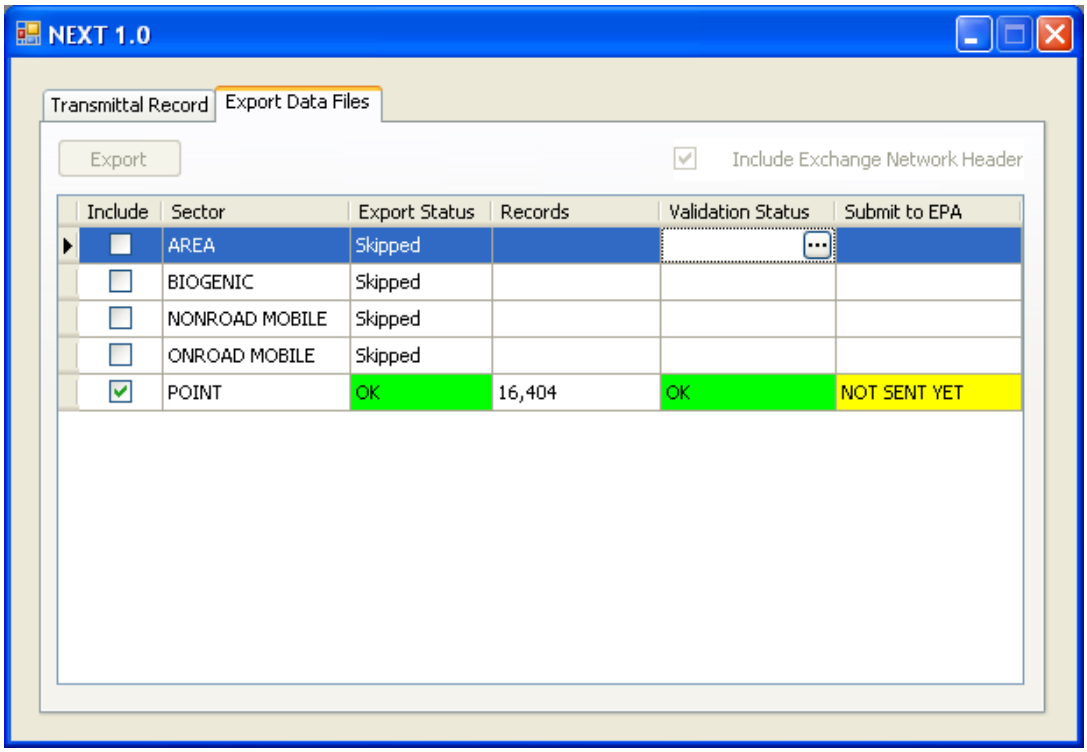
Wysocki, R. K., & McGary, R. (2003). *Effective Project Management: Traditional, Adaptive, Extreme* (3rd ed.). New York: Wiley.

Appendix 1. Core AirTools NEI tables Entity Relationship Diagram (ERD)



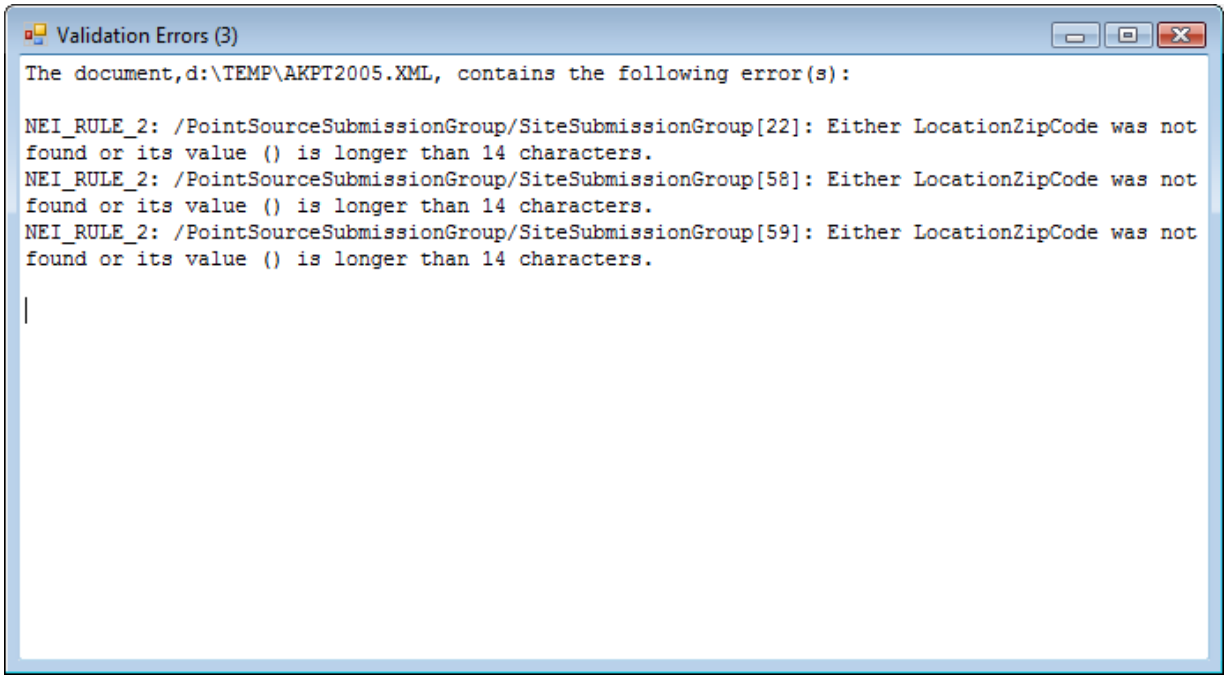
Appendix 2. Screenshot of NEXT form tabs



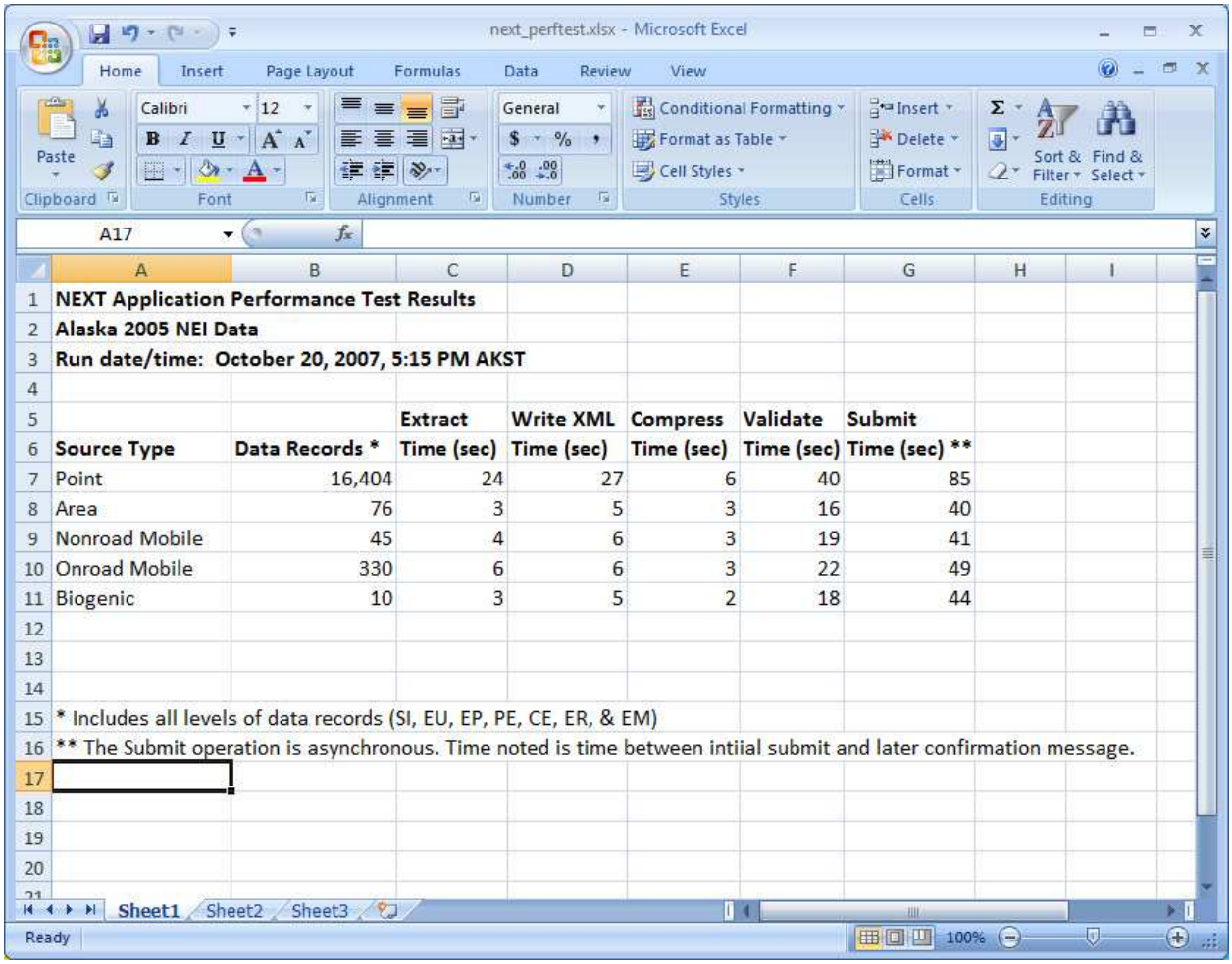


Appendix 3. Screenshot of successful 2005 point source NEI data submit log

Appendix 4. Screenshot of example failed point source NEI data validation log



Appendix 5. Screenshot of performance testing results spreadsheet



next_perftest.xlsx - Microsoft Excel

	A	B	C	D	E	F	G
1	**NEXT Application Performance Test Results**						
2	**Alaska 2005 NEI Data**						
3	**Run date/time: October 20, 2007, 5:15 PM AKST**						
4							
5			**Extract**	**Write XML**	**Compress**	**Validate**	**Submit**
6	**Source Type**	**Data Records ***	**Time (sec)**	**Time (sec)**	**Time (sec)**	**Time (sec)**	**Time (sec) ****
7	Point	16,404	24	27	6	40	85
8	Area	76	3	5	3	16	40
9	Nonroad Mobile	45	4	6	3	19	41
10	Onroad Mobile	330	6	6	3	22	49
11	Biogenic	10	3	5	2	18	44
12							
13							
14							
15	* Includes all levels of data records (SI, EU, EP, PE, CE, ER, & EM)						
16	** The Submit operation is asynchronous. Time noted is time between intiial submit and later confirmation message.						
17							
18							
19							
20							

Appendix 6. Screenshot of NUnit tests